Available online at www.sciencedirect.com
ScienceDirect
Computers & Operations Research 36 (2009) 167–175
computers & operations research
www.elsevier.com/locate/cor

A genetic algorithm for joint replenishment based on the exact inventory cost☆

Sung-Pil Hong[a], Yong-Hyuk Kim[b,*]

[a]*Department of Industrial Engineering, Seoul National University, Sillim-dong, Gwanak-gu, Seoul 151-742, Korea*
[b]*Department of Computer Science, Kwangwoon University, Wolgye-dong, Nowon-gu, Seoul 139-701, Korea*

Available online 1 September 2007

Abstract

Given the order cycles of items in joint replenishment, no closed-form formula or efficient method is known to compute the exact inventory cost. Previous studies avoid the difficulty by restricting the replenishment policy to the cases where the order cycle of each item is a multiple of the cycle of the most frequently ordered item. This simplifies the computation but may entail sub-optimality of a solution. To cope with this, we devise an unbiased estimator of the exact cost which is computable in time polynomial of the problem input size and $1/\varepsilon$, where ε is a pre-specified relative error of estimation. We then develop a genetic algorithm based on this new cost evaluation, report the experimental results in comparison to the "RAND" [Kaspi M, Rosenblatt MJ. An improvement of Silver's algorithm for the joint replenishment problem. IIE Transactions 1983; 15: 264–9] which has been known as a state-of-the-art method for joint replenishment, and discuss their implications.

Keywords: Inventory control; Multi-item; Joint replenishment problem; Approximation; Genetic algorithm

1. Introduction

Consider the following multiple-item extension of the economic order quantity (EOQ) model. There are m items and the ith item has demand λ_i per unit time, holding cost h_i per unit per unit time. Also, we assume two fixed order costs. Whenever an order is placed, it incurs the cost of K regardless of the combination of items and their quantity. If the ith item is included, we have to pay an additional cost of K_i independent of the order quantity. As in the basic EOQ model, stock-outs are not allowed. The problem is to find the order policy of the items that minimizes the average inventory cost per unit time. This model is called the *joint replenishment problem*, or JRP [1]. JRP has many potential applications in supply chain which typically involves multiple items. Indeed, an intensive literature has been accumulated on the model. The first method of JRP, to the author's best knowledge, is due to Goyal [2] (also see [3]). Silver proposed a simple but efficient method for JRP [4,5] which was improved by Kaspi and Rosenblatt to a heuristic called "RAND" reported to perform better than other methods available in the literature [6]. The RAND was modified and improved further in subsequent works (e.g., [7]).

☆ This work was partially supported by Research Fund for the new faculty of SNU.
* Corresponding author. Tel.: +82 2 940 5212.
E-mail address: yhdfly@kw.ac.kr (Y.-H. Kim).

0305-0548/$ - see front matter © 2007 Elsevier Ltd. All rights reserved.
doi:10.1016/j.cor.2007.08.006

To the best of the author's knowledge, there are three genetic algorithms (GAs) available in the literature. The first GA was due to Khousa et al. [8]. It is a generational pure GA and its performance was compared to the RAND algorithm [7]. Among 1600 instances, it outperformed RAND only in 12 instances, produced the solutions of the same quality in 862, and in the remaining 726 instances, it performed worse than RAND.

Olsen also proposed a GA for JRP [9]. This algorithm first decides grouping, namely, which items are replenished together and then assigns a fixed replenishment cycle for the items in the same group. The paper also identifies some problem parameters for which the GA outperforms the RAND. But, only in 5.5% of the instances, it obtained better solutions than RAND. Moon and Cha [10] proposed a GA for an extension of JRP with resource restriction. They also compared the GA with the RAND modified for the resource restriction. Again, the RAND outperformed the GA in most instances.

Although this paper deals with the basic JRP model described in the beginning of this section, we note that some recent studies considered the JRP models extended with additional constraints (e.g., budget or capacity constraints [10,11], and minimum order quantities [12]) or with non-deterministic demand [13,14].

2. Rationalization of JRP

Apparently, as a more general problem, JRP is harder than EOQ model. Indeed it is fundamentally harder: unlike the EOQ case, there are no crucial properties known on the optimality structures. For instance, it is not even clear if there is an optimal solution whose orders are collectively periodic, let alone stationary (i.e., each item is ordered with a fixed cycle). Due to this difficulty, JRP is normally modified with the following simplifying but perhaps practical assumptions.

Stationary orders: First, each item is replenished at equal time intervals. Hence, a replenishment policy is completely described by the vector of cycles of the m items: $T = (T_1, T_2, \ldots, T_m)$.

Synchronized rational cycles: The orders of items are synchronized over rational time units. In other words, there is a rational number $s \in \mathbb{Q}_+$ such that for all $i = 1, 2, \ldots, m$, $T_i = \tau_i s$ for some $\tau_i \in \mathbb{N}$.

Fig. 1 illustrates m stationary orders synchronized over the time epochs that are multiples of a fixed time unit s.

Under this simplification, s as well as $\tau = (\tau_1, \tau_2, \ldots, \tau_m)$ is a decision variable as it effects the cost. However, we note that even this simplified version of JRP is NP-hard when the time-horizon is finite [15]. From now on, by JRP we mean the JRP satisfying these assumptions. For simplicity, we assume that the unit time demands of all items are 2. We can do so, without loss of generality, by replacing each holding cost h_i by $2h_i/\lambda_i$. Then, besides the fixed cost K, the average inventory cost of each item i per unit time during its order cycle $T_i = \tau_i s$ is given as follows:

$$\frac{K_i}{\tau_i s} + h_i \tau_i s. \tag{1}$$

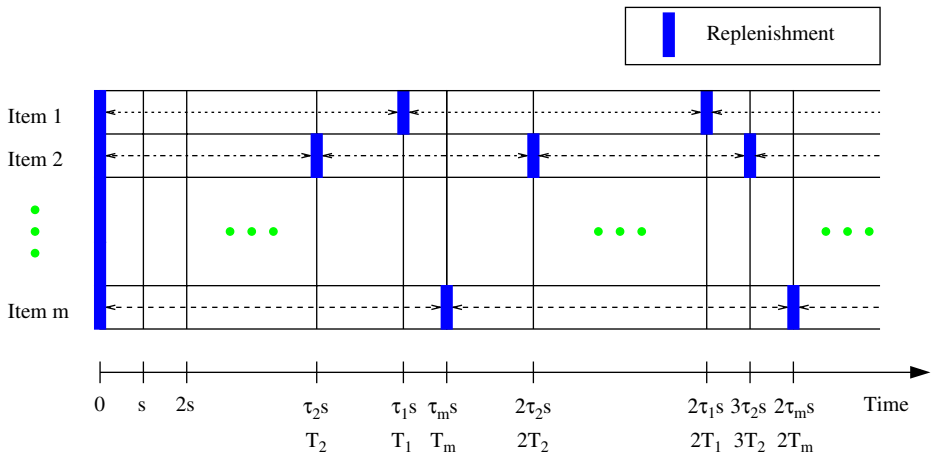

Fig. 1. An example of stationary orders synchronized on rational cycles.

Denote $M=\{\tau_1, \tau_2, \ldots, \tau_m\}$ and by $\mathrm{lcm}(S)$ the least common multiple of the numbers from $S \subseteq M$. Note that given S, $\mathrm{lcm}(S)$ can be computed efficiently. (See, e.g., [16, p. 861].) The orders of items are collectively repeated with the length of period $L=\mathrm{lcm}(M)$. Then, for $S \subseteq M$, $L/\mathrm{lcm}(S)$ is the number of time epochs from $t \in [0, L) := \{0, 1, 2, \ldots, L-1\}$ at which the items of S are simultaneously replenished. Then, from the inclusion–exclusion principle, the total number of distinct time epochs from $t \in [0, L)$ at which at least one item is replenished (and hence the fixed cost K incurs) is

$$\phi(\tau) = \sum_{i=1}^{|M|} \sum_{S \subseteq M:|S|=i} (-1)^{i-1} \frac{L}{\mathrm{lcm(S)}}. \tag{2}$$

Using this notation, given the order policy s and τ, the average inventory cost of the JRP is

$$G(\tau, s) = \frac{\phi(\tau)}{L}\frac{K}{s} + \sum_{i=1}^{m}\left(\frac{K_i}{\tau_i s} + h_i \tau_i s\right). \tag{3}$$

Formula (2) has exponentially many terms and, therefore, does not give us an acceptable computation method. In fact, no efficient computation of $\phi(\tau)$ is known in literature. We conjecture that computation of $\phi(\tau)$ is $\sharp P$-hard.

Virtually all the previous studies mentioned in Section 1 avoid this difficulty by introducing an additional assumption.

Multiple of minimum cycle: Each order cycle is a multiple of the minimum cycle of an item. With this restriction, the average inventory cost is reduced to

$$H(\tau, s) = \frac{K}{s} + \sum_{i=1}^{m}\left(\frac{K_i}{\tau_i s} + h_i \tau_i s\right). \tag{4}$$

Here, s is the cycle of the most frequently ordered item. Notice that once s is fixed, the problem reduces to the independent discrete-time EOQ problems of items. Most existing heuristics mentioned in Section 1 critically rely on this property. This simplifies the computation but may entail sub-optimality of a solution. We can construct such a 2-item example: $\lambda_1 = 800$, $\lambda_2 = 600$, $h_1 = 30$, $h_2 = 60$, $K_1 = 1500$, $K_2 = 1000$, and $K = 100$. It is not difficult to show that the optimal solution satisfying the three assumptions is $\tau = (2, 1)$ whose average cost is 17629.52. Without the last assumption, however, we can construct a better solution, $\tau = (3, 2)$, whose, objective value is 17527.12.

In this paper, we develop an algorithm for JRP that reflects the exact average cost given in (3). Although the exact and efficient computation of $\phi(\tau)$ is not known, we can, as will be shown in Section 3, instead approximate it in an efficient manner, namely, in time polynomial of the input size of the problem and the inverse of the approximation error. Furthermore, the approximation is an unbiased estimate of the exact cost. Then we will propose a GA utilizing this unbiased estimation of the exact cost.

3. Unbiased approximate counting of $\phi(\tau)$

The idea is random sampling time epochs to estimate the number of replenishment epochs among $[0, L)$. It is crucial to be done in an efficient manner. The following definition captures a concept of efficient counting.

Definition 3.1. Suppose we are given τ and $\varepsilon > 0$. An algorithm $\mathscr{A}$ is called a *fully polynomial time randomized approximation scheme* (FPRAS) [17] for counting $\phi(\tau)$ if $\mathscr{A}$ outputs Y, satisfying the following condition in time polynomial in the encoding length of τ and $1/\varepsilon$:

$$\Pr[|Y - \phi(\tau)| \leqslant \varepsilon\phi(\tau)] \geqslant \tfrac{3}{4}.$$

The principle of the FPRAS for counting the truth assignments satisfying a given disjunctive normal form [18] can be used to devise an FPRAS for counting $\phi(\tau)$. To do so, we first introduce some notations. Let S_i be the set of time epochs from $[0, L)$ at which the item i is replenished. Then $|S_i| = L/\tau_i$ and $\phi(\tau) = |\bigcup_{i=1}^{m} S_i|$. Also, for each $t \in [0, L)$ we denote as $r(t)$ the set of items replenished at time epoch t. Finally, let V be the union of S_i counting multiplicity. Thus $|V| = \sum_{i=1}^{m} |S_i| = \sum_{i=1}^{m} L/\tau_i$. Notice that V contains t, $r(t)$ number of times.

Consider the following two-phase random sampling. To each replenishment epoch t, assign the random variable $X(t) := |V|/r(t)$. In the first phase, select an item i with probability $|S_i|/|V|$. Then, choose a time epoch t from S_i equally likely.

Lemma 3.2. *X is an unbiased estimator of* $\phi(\tau)$: $E(X) = \phi(\tau)$.

Proof. We have

$$\Pr[t \text{ is chosen}] = \sum_{i=1}^{m} \Pr[t \text{ is chosen}|\text{item } i \text{ is chosen}] \Pr[\text{item } i \text{ is chosen}] = \sum_{i=1}^{m} \frac{1}{|S_i|} \frac{|S_i|}{|V|} = \frac{r(t)}{|V|}.$$

Therefore,

$$E(X) = \sum_{t} \Pr[t \text{ is chosen}] \cdot X(t) = \sum_{\text{replenishment epochs } t \in [0,L)} \frac{r(t)}{|V|} \frac{|V|}{r(t)} = \phi(\tau). \qquad \square$$

For each replenishment epoch t, we have $1 \leqslant r(t) \leqslant m$. Hence, $|V|/m \leqslant X(t) \leqslant |V|$ and $E(X) \geqslant |V|/m$. This also implies $|X(t) - E(X)| \leqslant (m-1)|V|/m$. The standard deviation is then $\sigma(X) \leqslant (m-1)|V|/m$. Combining this with $E(X) \geqslant |V|/m$, we get

$$\sigma(X) \leqslant (m-1)E(X). \tag{5}$$

Lemma 3.3. *Let* Y_k *be the average of k samples of* $X(t)$. *Then, if* $k = 4(m-1)^2/\varepsilon^2$, *then we have*

$$\Pr[|Y_k - \phi(\tau)| \leqslant \varepsilon\phi(\tau)] \geqslant \tfrac{3}{4}.$$

Proof. Using Chebyshev's inequality and $\sigma(Y_k) = \sigma(X)/\sqrt{k}$, we get,

$$\Pr[|Y_k - E(Y_k)| \geqslant \varepsilon E(Y_k)] \leqslant \left(\frac{\sigma(Y_k)}{\varepsilon E(Y_k)}\right)^2 = \left(\frac{\sigma(X)}{\varepsilon\sqrt{k}E(X)}\right)^2 \leqslant \frac{1}{4}.$$

The last inequality follows from (5). Since $E(Y_k) = E(X)$ which is, in turn, equal to $\phi(\tau)$ from Lemma 3.2, the lemma follows. $\square$

Theorem 3.4. *The two-phase sampling repeated* $k = 4(m-1)^2/\varepsilon^2$ *number of times yields an FPRAS for counting* $\phi(\tau)$.

Proof. Notice that $k = 4(m-1)^2/\varepsilon^2$ is bounded by a polynomial of the input size and $1/\varepsilon$. Hence, from Lemma 3.3 and Definition 3.1 the two-phase sampling yields an FPRAS for counting $\phi(\tau)$. $\square$

4. A GA

In this section we present the GA for JRP with two novel features. One is that each solution is evaluated by the estimator of the exact cost (3) provided by the FPRAS from Section 3. The other is its hybridization with a local search described below.

A GA [19] is a problem-solving technique motivated by Darwin's theory of natural selection in evolution. A basic framework of GAs was established by Holland [20]. At each iteration, a *generational* GA generates a considerable number of offspring per iteration while a *steady-state* GA generates a single offspring. A steady-state GA converges faster than a generational one at the expense of the diversity of population. If we add a local improvement typically after the mutation step, the GA is called a *hybrid* GA [21]. Fig. 2 shows a typical hybrid steady-state GA.

4.1. Genetic operators

The proposed GA is a standard hybrid steady-state GA which can be described as follows:

Representation: In our problem, a solution corresponds to an integer vector $\tau = (\tau_1, \tau_2, \ldots, \tau_m)$. Each solution in the population is represented by a *chromosome* which, in turn, consists of m *gene*s representing τ_i's. Hence an integer encoding is used instead of a binary encoding traditionally used in GA.

```
create an initial population of a fixed size;
repeat
        choose parent 1 and parent 2 from population;
        offspring ← crossover (parent 1, parent 2);
        mutation (offspring);
        local-improvement (offspring);
        replace (population, offspring);
until stopping condition is satisfied
return the best individual;
```

Fig. 2. A typical hybrid steady-state genetic algorithm.

```
// given τ
σ = (σ1, σ2, . . . , σm) ← a random permutation with length m;
for each i = 1, 2, . . . , m do
        if τσi - 1 is nonzero and better than τσi then
                repeat
                        τσi ← τσi - 1;
                until there is no improvement
        end if
        if τσi + 1 is better than τσi then
                repeat
                        τσi ← τσi + 1;
                until there is no improvement
        end if
end for
return τ = (τ1, τ2, . . . ,τm);
```

Fig. 3. A local improvement algorithm.

Initial population: The GA first creates p integer vectors at random. Each gene is set to be a random integer value between 1 and 10.

Selection: We assign to each solution in the population a fitness value calculated from its objective function value. Each chromosome is selected as a parent with a probability that is proportional to its fitness value. In selecting two parents, we use the very common parent selection scheme called *roulette-wheel-based proportional selection* scheme where the probability for the best solution to be chosen is four times higher than that for the worst solution. The fitness value f_i of the i-th chromosome can be represented as follows: $f_i = (c_w - c_i) + (c_w - c_b)/3$, where c_b, c_w, and c_i are the objective function values of the best chromosome, the worst chromosome, and the ith chromosome, respectively.

Crossover and mutation: Crossover and mutation are the two most important space exploration operators in GAs. A crossover operator creates a new offspring by combining parts of the parents. We use the most popular *one-point crossover* as used in [8]. It randomly selects a cut point which is the same on both parent chromosomes. The cut point divides the chromosome into two disjoint parts: the left part and the right one. The left part of parent 1 is copied to the same locations of the offspring chromosome. Similarly, the right part of parent 2 is copied to the same locations of the offspring chromosome. That is, the crossover creates an offspring by gluing parts of the parent chromosomes.

After a crossover, mutation operator is applied to the offspring. Mutation changes a gene in a chromosome with a fixed (and relatively small) probability p_m. Each gene is mutated over the integers from 1 to 10 with the probability p_m.

Local improvement: After crossover and mutation, the proposed GA applies a local improvement process on the offspring. GAs are known to be not so good at fine tuning around local optima. Pure GAs usually take fairly many iterations but the performance is still not so desirable in many cases apparently because of this weakness. Thus, a local improvement heuristic is applied on a new offspring to fine tune it. We use a simple local improvement heuristic described in Fig. 3. It improves upon a given integer vector τ by decreasing and increasing τ_i for each i until no improvement can be obtained.

Replacement: After having generated a new offspring and trying to locally improve it, the proposed GA replaces a member of the population with the new offspring. We use a replacement scheme, called *preselection* [22], in which an offspring replaces the inferior parent (only when the offspring is better) hoping to maintain population diversity.

Stopping condition: Many GAs still run for a fixed number of generations before stopping. One of the usually better criteria for stopping is to stop when the population diversity drops below a certain threshold. We use a combined condition for stopping. The proposed GA terminates when one of the two conditions is satisfied: (i) the number of generations reaches N_g or (ii) the fitness of the worst chromosome is equal to that of the best one (and hence all chromosomes of the population have the same quality).

Objective functions: The evaluation process is applied to every chromosome generated during the initialization. As the GA progresses through the generations, it is not necessary to reevaluate the entire population. However, when new chromosomes (offspring) are created, they should be evaluated.

To find out the effects of the exact cost and local improvement separately, we will compare two new GAs: *HGA-old* and *HGA-new*. In HGA-old, we use the cost function (4) as the previous studies in evaluating a solution. In HGA-new, a solution is evaluated based on the estimator of the exact cost (3) produced by FPRAS. We set $\varepsilon = 0.1$. If L is less than or equal to 1,000,000, we use the exact cost (3) by using the straightforward counting of replenishment epochs from $[0, L)$ rather than relying on FPRAS estimator.

Parameter setting: We set the population size p to be 50. The mutation probability p_m was set to 0.1. The maximum number of generations (N_g) was set to 2000. From a preliminary test, we chose the best parameter setting among $\{p\} = \{30, 50, 100\}$, $\{p_m\} = \{0.05, 0.1, 0.2\}$, and $\{N_g\} = \{1000, 1500, 2000, 3000\}$.

5. Experimental results and discussion

We compare four methods: the Goyal's method [23], RAND [6], HGA-old, and HGA-new. The test instances were designed after the experiments in [7,9,24]. Each instance consists of five parameters summarized in Table 1. We consider 540($=4 \times 5 \times 3 \times 3 \times 3$) combinations of the parameters. As 30 instances were generated for each combination, we had the total of 16,200 instances.

As the evaluation of HGA-new is based on the FPRAS estimator of the exact inventory cost, we want to check first if the cost of HGA-new solutions reflects the true cost. To see this we apply HGA-new to all 4050 instances corresponding to $m = 10$. In doing so, we always use FPRAS in evaluation (even when $L \leqslant 1,000,000$), and compute the exact cost of the final solutions. The exact cost computation is done by the straightforward counting of replenishment epochs from $[0, L)$. (In fact, $m = 10$ was the largest number of such exact cost computing that is possible.) Table 2 summarizes the relative errors of mean estimated to mean exact cost of the HGA-new solutions. This shows that FPRAS provides the estimators which is very close to the real values.

Table 1
Instance parameters

Parameters	Ranges
# of items m	5, 10, 20, 30
Fixed cost K	50, 200, 500, 2000, 5000
Fixed cost K_i	[50, 300], [400, 1000], [1100, 2000]
Demand λ_i	[500, 1000], [300, 1500], [200, 2000]
Holding cost h_i	[20, 200], [200, 500], [500, 1000]

There are 540 instance groups and 30 instances are generated for each group.

Table 2
Relative error of estimated to exact cost of HGA-new solutions

K	50	200	500	2000	5000
Estimation	206527.7895	210476.9155	217307.0641	242376.4146	280346.3566
Exact	206527.7936	210476.9155	217307.0641	242376.4146	280346.3566
Rel. error (%)	0.00001	0.00000	0.00000	0.00000	0.00000

Table 3
Relative %-difference from RAND solutions and CPU time in seconds

m	K	RAND [7] Gap (CPU)	GOYAL [23] Gap (CPU)	HGA-old Gap (CPU)	HGA-new Gap (CPU)
5	50	−(0.001)	0.13848 (0.000)	−0.20943 (0.254)	−0.30210 (0.755)
	200	−(0.001)	0.04883 (0.000)	−0.00452 (0.191)	−0.05958 (0.310)
	500	−(0.001)	0.02087 (0.000)	0.00000 (0.120)	−0.00114 (0.203)
	2000	−(0.001)	0.00150 (0.000)	0.00000 (0.109)	0.00000 (0.131)
	5000	−(0.001)	0.00048 (0.000)	0.00000 (0.086)	0.00000 (0.117)
	Mean	−(0.001)	0.04203 (0.000)	−0.04279 (0.152)	−0.07256 (0.303)
10	50	−(0.001)	0.22106 (0.000)	−0.45220 (4.452)	−0.55068 (10.21)
	200	−(0.001)	0.08457 (0.000)	−0.10104 (1.323)	−0.21546 (1.957)
	500	−(0.001)	0.03019 (0.000)	0.00936 (0.865)	−0.03868 (1.345)
	2000	−(0.001)	0.00307 (0.000)	−0.00002 (0.531)	−0.00002 (0.682)
	5000	−(0.001)	0.00003 (0.000)	0.00000 (0.510)	0.00000 (0.543)
	Mean	−(0.001)	0.06778 (0.000)	−0.10878 (1.536)	−0.16097 (2.947)
20	50	−(0.003)	0.19629 (0.001)	−0.68861 (29.30)	−0.79604 (59.27)
	200	−(0.003)	0.08571 (0.001)	−0.31599 (19.97)	−0.38099 (22.24)
	500	−(0.003)	0.04814 (0.001)	−0.03522 (10.76)	−0.15720 (10.32)
	2000	−(0.003)	0.00519 (0.001)	0.00594 (3.866)	0.03868 (5.583)
	5000	−(0.003)	0.00055 (0.001)	−0.00013 (3.203)	−0.00013 (3.110)
	Mean	−(0.003)	0.06717 (0.001)	−0.20680 (13.42)	−0.25914 (20.11)
30	50	−(0.006)	0.29265 (0.001)	−0.80668 (213.3)	−0.87693 (457.6)
	200	−(0.006)	0.09751 (0.001)	−0.45375 (26.33)	−0.54639 (51.42)
	500	−(0.006)	0.04064 (0.001)	−0.13168 (24.19)	−0.27195 (32.25)
	2000	−(0.006)	0.00392 (0.001)	0.01189 (14.40)	0.00788 (19.43)
	5000	−(0.006)	0.00160 (0.001)	−0.00038 (11.30)	−0.00040 (13.07)
	Mean	−(0.006)	0.08726 (0.001)	−0.27612 (57.90)	−0.33756 (114.8)

Each cell contains two values: the average value of $\frac{100\times(Method\text{'s cost}-\text{RAND's cost})}{\text{RAND's cost}}$, and CPU seconds on Intel Xeon CPU 2.4 GHz. Each cell corresponds to 810 instances.

Each cell of Table 3 contains two values: the mean percentage relative difference of the method's solution cost from the RAND's solution cost: 100× (*Method's cost* - RAND's cost)/RAND's cost, and CPU seconds on Intel Xeon CPU 2.4 GHz.

In sum, the proposed GAs provide better solutions at the expense of computational time.

With a few exceptions, $(m, K) = (10, 500)$, $(20, 2000)$, and $(30, 2000)$, HGA-old provides the solutions equal to or of better quality than RAND. And the margins tend to get larger as the K gets smaller. Thus, our GA is more effective when the fixed costs of items, K_i, are comparable to K. On the other hand, HGA-new outperforms RAND except the cases $(m, K) = (20, 2000)$ and $(30, 2000)$. As expected from the evaluation based on the exact cost, it performs even better than HGA-old.

Notice that each cell of Table 3 represents the mean value of 810 instances to a combination of K and m. Table 4 indicates for how many of the 810 instances, the corresponding methods perform better than RAND or vice versa. In most combinations of m and K, GAs perform better than RAND.

From these, we can see that both the local improvement and the exact cost-based evaluation are effective in pursuing solutions of better objective values.

However, Table 3 shows that HGA-new took almost 8 min for the instances with $m = 30$ and $K = 50$. Although it seems an acceptable time for the practical purpose of JRP, which does not require a real-time solution, it is several orders of magnitude larger than those for RAND or the Goyal's method. Also we note that from a preliminary test, HGA-new took more than 30 min for some instances with $m = 50$ and $K = 50$. This intensive computational effort is primarily due to the operations on large numbers arising, for instance, from the least common multiples of τ_i's.

In this sense, an improved solution at the expense of computation time is not the only implication of this work especially considering the very cheap computation times of the Goyal's method and RAND. Actually, a motivation of our work is to see how good solutions do these cheap methods provide compared to an elaborated method such

Table 4
Percentage of better/worse instances

Method		GOYAL [23]		HGA-old		HGA-new	
m	*K*	Better	Worse	Better	Worse	Better	Worse
5	50	1.73	15.19	30.49	0.12	59.88	0.00
	200	0.62	7.04	2.10	0.00	19.38	0.00
	500	0.00	3.46	0.00	0.00	0.86	0.00
	2000	0.00	0.74	0.00	0.00	0.00	0.00
	5000	0.00	0.12	0.00	0.00	0.00	0.00
	Mean	0.47	5.31	6.52	0.02	16.02	0.00
10	50	1.48	29.63	67.53	2.96	83.58	0.12
	200	0.12	16.67	26.05	6.42	51.36	0.00
	500	0.12	8.77	1.85	2.72	14.81	0.12
	2000	0.12	1.23	0.12	0.00	0.12	0.00
	5000	0.00	0.12	0.00	0.00	0.00	0.00
	Mean	0.37	11.28	19.11	2.42	29.98	0.05
20	50	3.70	34.32	87.28	4.94	96.91	0.86
	200	3.58	23.95	59.63	5.43	70.99	11.48
	500	2.47	14.20	23.09	11.48	48.40	10.86
	2000	1.73	4.07	1.73	0.86	2.22	9.51
	5000	1.85	1.85	1.85	0.00	1.85	0.00
	Mean	2.67	15.68	34.72	4.54	44.07	6.54
30	50	7.53	46.05	95.31	1.98	99.01	0.86
	200	7.53	31.23	72.47	4.94	85.43	5.19
	500	7.53	17.04	39.51	9.38	65.56	3.70
	2000	7.16	5.93	7.16	2.47	13.33	9.01
	5000	4.69	2.96	5.31	0.12	5.31	0.12
	Mean	6.89	20.64	43.95	3.78	53.73	3.78

For each combination of *m* and *K*, there are 810 instances.

as the GA based on the exact cost evaluation, and hybridization with a local search. Notice that the new GAs and the previous methods, the Goyal's method and RAND, have a very marginal difference in objective values. This has been typically observed also in the literature discussed in Section 1. This is also reminiscent of the fact that the basic EOQ inventory cost is quite insensitive to the change in order quantity (see, e.g., [25, p. 208]) or that the restriction of replenishment policy to very special type of solutions guarantees solutions within 2% of the exact optimum for the single-warehouse-multi-retailer problem [26].

Overall, although JRP is an optimization problem whose general optimal structure and computational complexity are largely unexplored in theoretical sense, it does not seem, after all, a difficult problem in practice. We conjecture that the Goyal's method and RAND as well as HGA-new and HGA-old provide the policies which are no more than a few percents off the global optimum.

References

[1] Goyal SK. Determination of economic packaging frequency for items jointly replenished. Management Science 1973;20:232–8.
[2] Goyal SK. Determination of optimum packaging frequency of items jointly replenished. Management Science 1974;21:436–43.
[3] van Eijs MJG. A note on the joint replenishment problem under constant demand. Journal of Operational Research Society 1993;44:185–91.
[4] Silver EA. Modifying the economic order quantity (EOQ) to handle coordinated replenishment of two or more items. Production and Inventory Management 1975;16:26–38.
[5] Silver EA. A simple method of determining order quantities in joint replenishments under deterministic demand. Management Science 1976;22:1351–61.
[6] Kaspi M, Rosenblatt MJ. An improvement of Silver's algorithm for the joint replenishment problem. IIE Transactions 1983;15:264–9.
[7] Kaspi M, Rosenblatt MJ. On the economic ordering quantity for jointly replenished items. International Journal of Production Research 1991;29:107–14.

[8] Khousa M, Michalewicz Z, Satoskar SS. A comparison between genetic algorithms and the RAND method for solving the joint replenishment problem. Production Planning & Control 2000;11:556–64.
[9] Olsen AL. An evolutionary algorithm to solve the joint replenishment problem using direct grouping. Computers & Industrial Engineering 2005;48:223–35.
[10] Moon IK, Cha BC. The joint replenishment problem with resource restriction. European Journal of Operational Research 2006;173:190–8.
[11] Hoque MA. An optimal solution technique for the joint replenishment problem with storage and transport capacities and budget constraints. European Journal of Operational Research 2006;175:1033–42.
[12] Porras E, Dekker R. An efficient optimal solution method for the joint replenishment problem with minimum order quantities. European Journal of Operational Research 2006;174:1595–615.
[13] Lee LH, Chew EP. A dynamic joint replenishment policy with auto-correlated demand. European Journal of Operational Research 2005;165: 729–47.
[14] Nielsen C, Larsen C. An analytical study of the $Q(s, S)$ policy applied to the joint replenishment problem. European Journal of Operational Research 2005;163:721–32.
[15] Arkin E, Joneja D, Roundy R. Computational complexity of uncapacitated multi-echelon production planning problems. Operations Research Letters 1989;8:61–6.
[16] Cormen TH, Leiserson CE, Rivest RL, Stein C. Introduction to algorithms. 2nd ed., Cambridge, MA: MIT Press; 2001.
[17] Karp RM, Luby M, Madras N. Monte Carlo approximation algorithms for enumeration problems. Journal of Algorithms 1983;10:429–48.
[18] Karp RM, Luby M, Monte Carlo algorithms for enumeration and reliability problems. In: Proceedings of the 24th IEEE annual symposium on foundations of computer science; 1983, p. 56–64.
[19] Whitley D. A genetic algorithm tutorial. Statistics and Computing 1994;4:65–85.
[20] Holland J. Adaption in Natural and Artificial Systems. Ann Arbor: University of Michigan Press; 1975.
[21] Bui TN, Moon BR. Genetic algorithm and graph partitioning. IEEE Transactions on Computers 1996;45:841–55.
[22] D. Cavicchio, Adaptive search using simulated evolution, Ph.D. thesis, University of Michigan, Ann Arbor; 1970.
[23] Goyal SK. Optimum ordering policy for a multi item single supplier system. Operational Research Quarterly 1974;25:293–8.
[24] Goyal SK, Deshmukh SG. A note on 'the economic ordering quantity for jointly replenished items'. International Journal of Production Research 1993;31:109–16.
[25] Nahmias S. Production and operations analysis. 4th ed., New York: McGraw-Hill; 2001.
[26] Roundy RO. 98%-effective integer-ratio lot-sizing for one-warehous multi-retailer systems. Management Science 1985;31:1416–30.